In the Specification

Please amend the specification of this application as follows:

Rewrite paragraph [01] as follows:

--[01] This application claims priority under 35 USC §119(e)(1) of Provisional Application Serial No. 60/263,804, filed January 24, 2001 ~~(TI-31599PS)~~ and Provisional Application Serial No. 60/315,748 ~~(TI-33109P)~~ filed August 29, 2001.--

Rewrite paragraph [02] as follows:

--[02] This application is related to and claims priority under 35 USC §119 (e)(1) to Provisional Application Serial No. 60/263,804, ~~(TI-31599P)~~ *Host Software Assisted Transparent Shared Memory Support For Multiple CPU Embedded Development Systems*, filed on January 24, 2001. This application is also related to co-pending applications Serial No. 60/315,847 ~~(TI-31599P)~~ *Transparent Shared Memory Access in a Software Development System* filed August 29, 2001 now U.S. Patent Application Serial No. 09/998,755 filed December 3, 2001, Serial No. 60/315,815 ~~(TI-33242P)~~ *Method for Maintaining Cache Coherency in Software in a Shared Memory System* filed August 29, 2001 now U.S. Patent Application Serial No. 09/998,330 filed December 3, 2001, and Serial No. 60/315,843 ~~(TI-33316P)~~ *Software Shared Memory Bus* filed August 29, 2001 now U.S. Patent Application Serial No. 09/998,329 filed December 3, 2001.--

Rewrite paragraph [07] as follows:

--[07] In other embodiments, the method includes a step of creating a software memory map representing the memory usage of the processors in the multiple processor system.

When a software breakpoint is to be set in a shared memory location, the software memory map is searched to ~~located~~ locate all processors having read access to that shared memory location. The

software representation for software breakpoints maintained for the located processors is updated to reflect that software breakpoint is being set at the shared memory location. Then, the software breakpoint instruction is written to the shared memory location. When a software breakpoint is cleared in ~~shard~~ shared memory, the original instruction stored in a software representation maintained for software breakpoints is written into the shared memory location. The software memory map is ~~search~~ searched to find all processors having read access to the shared memory location and the software breakpoint representation is updated for each located processor to reflect the removal of the software breakpoint.--

Rewrite paragraph [09] as follows:

--[09] In another embodiment, the method ~~of claim one~~ is enhanced to maintain software breakpoint coherency when stepping over a breakpoint or running after hitting a breakpoint. The software breakpoint in the shared memory location is cleared such that all active debug sessions are notified that the breakpoint has been removed, the processor requesting the step over or resumption of execution is stepped to the instruction following the shared memory location, and the software breakpoint is reset in the shared memory location such that all active debug sessions are notified of the ~~setting~~ resetting of the breakpoint.--

Rewrite paragraph [46] as follows:

--[46] Figure 7 is a block diagram illustrating emulation logic 108 of Figure 1 in more detail. Emulation circuitry 108 provides common debug accesses (reading and writing of memory and registers) without direct CPU intervention through a Debug and Test Direct Memory Access (DT-DMA) mechanism 108a. Because the DT-DMA mechanism uses the same memory access mechanism as the CPU, any read or write access that the CPU can perform in a single operation can be done

via a DT-DMA memory access. The DT-DMA mechanism will present an address via address bus 720 (and data via interface 710, in the case of a write) to the CPU, which will perform the operation during an open bus cycle slot. DT-DMA request signal 721 is asserted by the emulation circuitry to request a read or write transaction. Once memory ~~812~~ 712 or ~~830~~ 730 has provided the desired data, it is presented back to the DT-DMA mechanism. DT-DMA ready signal 722 is asserted by instruction buffer unit 706 to indicate that a requested data item is available to the emulation circuitry.--

Rewrite paragraph [48] as follows:

--[48] For a data write, Data Address Generation circuitry (DAGEN) 722 schedules a write request in response to a request 721 from the emulation circuitry and a DT-DMA address is placed on the address bus EAB 752. Write data is simultaneously placed on the E and F busses 750 in response to a control signal. A tag signal on address bus EAB 752 is also asserted by DAGEN 722 in response to the DT-DMA request so that the write transaction can be identified as such by instruction cache 606, which monitors the write address bus EAB 752. Coherence circuitry 716 monitors address bus EAB 752 and causes cache 606 to invalidate a cache entry, if the address of a cache entry matches the DT-DMA write address.--

Rewrite paragraph [54] as follows:

--[54] GEL_MapAddStr() has four parameters: address, page, length, "attribute", and waitstate. The address parameter identifies the starting address of a range in memory. The page parameter identifies the type of memory in the address range: instruction memory, data memory, or I/O space. The length parameter defines the length of the range of memory. The attribute parameter is a string that defines one or more attributes for the specified

memory range. The waitstate parameter defines the number of waitstates for that memory range.--

Rewrite paragraph [67] as follows:

--[67] The software memory map is created and the debug sessions are activated as described with Figure 9 above. Bus manager 1106 of Figure 11 intercepts the software breakpoint setting and clearing requests from each active debug session 1104 and causes all debug sessions 1104 to be notified of any breakpoint changes in common shared memory. When a breakpoint is set or cleared in common shared memory by a debug session 1104, bus manager 1106 searches memory maps 1102 to locate all processors ~~601~~ 510 having read access to the common shared memory and their associated debug sessions. Bus manager 1106 interacts with the breakpoint manager 1110 of each located debug session 1104 to update the breakpoint table for the debug session appropriately.--